Two-Timescale Q-Learning with an Application to Routing in Communication Networks[1]

Mohan Babu K [2] and Shalabh Bhatnagar[3]†

Abstract

We propose two variants of the Q-learning algorithm that (both) use two timescales. One of these updates Q-values of all feasible state-action pairs at each instant while the other updates Q-values of states with actions chosen according to the 'current' randomized policy updates. A sketch of convergence of the algorithms is shown. Finally, numerical experiments using the proposed algorithms for routing on different network topologies are presented and performance comparisons with the regular Q-learning algorithm are shown.

Index Terms

Q-learning based algorithms, Markov decision processes, two-timescale stochastic approximation, simultaneous perturbation stochastic approximation (SPSA), normalized Hadamard matrices.

I. Introduction

Dynamic programming is a classical approach for solving decision making problems. However, traditional solution methodologies such as policy iteration and value iteration for solving the Bellman equation for optimality require complete knowledge of the system model. Moreover, the computational requirements for solving the Bellman equation become prohibitive in the presence of large state spaces. Motivated by these considerations, there has been significant research on simulation based methods for solving the Bellman equation, that largely go under

[1]This work was supported in part by Grant No. SR/S3/EE/43/2002-SERC-Engg from the Department of Science and Technology, Government of India.

[2] Department of Electrical Engineering, Indian Institute of Science, Bangalore 560 012, India. E-mail: mohan@ee.iisc.ernet.in

[3]Department of Computer Science and Automation, Indian Institute of Science, Bangalore 560 012, India. E-mail: shalabh@csa.iisc.ernet.in

†Corresponding author. E-Mail for correspondence: shalabh@csa.iisc.ernet.in

the name reinforcement learning or neuro-dynamic programming [2], [14]. The main idea here is to simulate transitions instead of computing transition probabilities that may be hard to obtain and to use parametric representations of the cost-to-go function. In [16], a simulation based variant of the value iteration algorithm that goes under the name Q-learning was proposed. Here one estimates certain Q-factors (or Q-values) via simulation. These are (cost related) values associated with each feasible state-action pair. The Q-learning algorithm has become a popular reinforcement learning technique and has been applied in several settings.

Efficiently routing data is a difficult and important decision making problem in communication networks. The problem becomes even more difficult if one takes into account the fact that networks are subject to frequent and unpredictable changes in topology and link costs. Conventional internet routing algorithms such as RIP or OSPF are based on minimizing the number of hops which means number of relay nodes between the source and destination. With every change in topology, the network generates several routing information packets and a lot of time is needed to arrive at optimal paths. Reinforcement learning based methods have recently been applied for routing in communication networks. For instance, [10] uses temporal difference learning for routing in integrated service networks. In [7], a 'Q-routing' algorithm based on Q-learning is proposed. An adaptive algorithm that uses dual reinforcement Q-learning is proposed in [9]. In [13], a reinforcement learning type algorithm based on certain 'biological ants' that explore the network and detect loops in the path is proposed.

In this note, we develop two Q-learning algorithms that (both) use two-timescale stochastic approximation. The first of these updates Q-values of all feasible state-action pairs at each instant while the other updates Q-values of states with actions chosen according to the 'current' randomized policy updates. Two-timescale stochastic approximation [6], [4] has been well studied; more recently, also in the context of actor-critic type reinforcement learning algorithms for Markov decision processes [8], [5]. The usual Q-learning algorithm involves the computation of an action that attains the minimum value amongst current Q-factor updates. The above minimum is obtained prior to the averaging step. Our algorithms obtain minimum in the space of stationary randomized policies using a gradient search recursion with simultaneous perturbation stochastic approximation (SPSA) [12] estimates on the faster timescale and average Q-factors on the slower timescale. Since we use stationary randomized policy updates for picking the 'minimizing actions' in the Q-value updates, other actions also get picked with certain (albeit diminishing)

probabilities obtained from the randomized policy updates. Hence we believe that our algorithms do not suffer from problems due to *lack of sufficient exploration* [14] associated with regular Q-learning. We perform numerical experiments using our algorithms on a few different settings and show performance comparisons with the Q-learning algorithm. Both our algorithms exhibit fast convergence and give the corresponding optimal policies. Our second algorithm may have computational advantages in scenarios where the number of feasible actions in each state is large.

The rest of the note is organized as follows. The framework and algorithms are presented in Section II. The convergence analysis of the algorithms is briefly sketched in Section III. Numerical experiments over different network topologies for a problem of routing are presented in Section IV. Finally, Section V presents the concluding remarks.

II. Framework and Algorithm

Suppose $\{X_n\}$ is a controlled Markov chain or a Markov decision Process (MDP) taking values in a set $S = \{1, 2, \ldots, s\}$ with $s < \infty$. Let $U(i)$ denote the set of all feasible actions or controls in state $i \in S$. We assume that $U(i)$ are finite sets and in particular have the form $U(i) = \{u_i^0, u_i^1, \ldots, u_i^N\}$. Note that we assume only for notational simplicity that each feasible action set $U(i)$ has exactly $(N+1)$ elements. In general, N could be a function of state i. Suppose $p(i,u,j)$ and $g(i,u,j), i,j \in S, u \in U(i)$, respectively, denote the one-step transition probability and the single-stage cost when the current state of the MDP is i, in which a feasible action u is chosen and the next state is j. We assume that $g(i,u,j)$ are nonnegative and bounded. Let $U = \bigcup_{i \in S} U(i)$ denote the set of all possible actions. By an admissible policy $\bar{\psi}$, we mean a sequence of functions $\bar{\psi} = \{\mu_0, \mu_1, \mu_2, \ldots\}$ with each $\mu_k : S \to U$ such that $\mu_k(i) \in U(i)$, $i \in S, k \geq 0$. Let Ψ be the set of all admissible policies. If $\mu_k = \mu, \forall k \geq 0$, then we call $\bar{\psi} = \{\mu, \mu, \ldots\}$ or by abuse of notation, the function μ itself as a stationary policy. Let $\alpha \in (0,1)$ be the discount factor. The aim is to minimize over all admissible policies $\bar{\psi} = \{\mu_0, \mu_1, \mu_2 \ldots\}$, the infinite horizon α-discounted cost

$$J_{\bar{\psi}}(i) = \lim_{T \to \infty} E\left[\sum_{k=0}^{T} \alpha^k g(X_k, \mu_k(X_k), X_{k+1}) | X_0 = i\right]. \tag{1}$$

Let

$$J^*(i) = \min_{\bar{\psi} \in \Psi} J_{\bar{\psi}}(i), i \in S \tag{2}$$

denote the optimal cost. For a given stationary policy μ, the function $J_\mu(.)$ is called the value function corresponding to policy μ. One can show that an optimal stationary policy exists for this problem and the optimal cost J^* satisfies the Bellman equation

$$J^*(i) = \min_{u \in U(i)} \left(\sum_{j \in S} p(i,u,j)(g(i,u,j) + \alpha J^*(j)) \right). \tag{3}$$

For given $i \in S, u \in U(i)$, let

$$Q^*(i,u) = \sum_{j \in S} p(i,u,j) \left(g(i,u,j) + \alpha J^*(j) \right). \tag{4}$$

Quantities $Q^*(i,u)$ are called the optimal Q-factors or Q-values [2]. The Bellman equation can now be written as

$$J^*(i) = \min_{u \in U(i)} Q^*(i,u), i \in S. \tag{5}$$

As a result of the above, (4) becomes

$$Q^*(i,u) = \sum_{j \in S} p(i,u,j) \left(g(i,u,j) + \alpha \min_{v \in U(j)} Q^*(j,v) \right). \tag{6}$$

The usual Q-learning algorithm is a stochastic approximation version of (6) and proceeds as follows:

$$Q_{n+1}(i,u) = Q_n(i,u) + \gamma(n) \left(g(i,u,\eta_n(i,u)) + \alpha \min_{v \in U(\eta_n(i,u))} Q_n(\eta_n(i,u),v) - Q_n(i,u) \right). \tag{7}$$

In the above, $\eta_n(i,u)$ is a simulated next state when current state is i and action $u \in U(i)$ is used. It is assumed that $\eta_n(i,u), n \geq 0$, are independent random variables each having distribution $p(i,u,.)$. Also, $\{\gamma(n)\}$ is a step-size sequence that satisfies $\gamma(n) > 0 \ \forall n \geq 0$,

$$\sum_n \gamma(n) = \infty \text{ and } \sum_n \gamma(n)^2 < \infty.$$

In what follows, we present two variants of (7) that both use two-timescale stochastic approximation. In these algorithms, explicit minimization in (7) is avoided and instead a gradient search over randomized policies is performed on a faster timescale. While the first variant updates Q-values for all state-action pairs at each instant, the second variant updates Q-values at each instant for pairs of states with actions picked according to current randomized policy updates. The second algorithm thus provides solution to the following system of equations instead of (4).

$$Q^*_{\pi_i}(i) = \sum_{u \in U(i)} \pi_i(u) \sum_{j \in S} p(i,u,j) \left(g(i,u,j) + \alpha \min_{\pi'_j} Q^*_{\pi'_j}(j) \right). \tag{8}$$

Note that in (8), Q-values are associated with state-randomized policy pairs rather than state-action pairs. Here $\pi_i = (\pi_i(u_i^0), \ldots, \pi_i(u_i^N))$ with $\pi_i(u)$ being the probability of picking action u in state i. Thus $\pi = (\pi_1, \ldots, \pi_s)$ represents a stationary randomized policy $\pi : S \rightarrow P(U)$ with $\pi_i \in P(U(i))$. Here for any set $A, P(A)$ denotes the set of all probability measures on A. In what follows, we shall represent a randomized policy using $\pi = (\hat{\pi}_1, \ldots, \hat{\pi}_s)$ with each $\hat{\pi}_i = (\pi_i(u), u \in U(i) \backslash \{u_i^0\})$. Note that this is a valid representation as $\pi_i(u_i^0)$, $i \in S$, are obtained from the components of $\hat{\pi}_i$ as $\pi_i(u_i^0) = 1 - \sum_{j=1}^N \pi_i(u_i^j)$. Let $PS \subset \mathcal{R}^N$ denote the simplex

$$PS = \{(y_1, \ldots, y_N) | y_i \geq 0, 1 \leq i \leq N \ and \ \sum_{i=1}^N y_i \leq 1\},$$

in which $\hat{\pi}_i, i \in S$, take values. Let $\Gamma : \mathcal{R}^N \rightarrow PS$ denote the projection map. Consider now $\{\pm 1\}^N$-valued vectors $\Delta_n(i) = \left(\Delta_n(i, u_i^1), \ldots, \Delta_n(i, u_i^N)\right), i \in S$. These are used for perturbing policy updates in order to obtain suitable gradient estimates of the Q-function. In two and one-simulation SPSA based gradient estimates [11], [12], perturbation variables such as $\Delta_n(i, u_i^j), n \geq 0, 1 \leq j \leq N$ are considered to be i.i.d., symmetric, mean-zero random variables. In the case of one-simulation SPSA algorithms [12], it was observed in the context of simulation optimization in [4] that performance is considerably improved by using a deterministic construction, based on certain normalized Hadamard matrices, for the perturbations $\Delta_n(i)$. We therefore use such a construction in what follows.

A. Construction for Perturbation Sequences $\Delta_n(i)$

As in [4], let H_P be a normalized Hadamard matrix (a Hadamard matrix is said to be normalized if all the elements of its first row and column are 1s) of order P with $P \geq N + 1$. Let $h(1), \ldots, h(N)$ be any N columns other than the first column of H_P, and form a new $P \times N$ dimensional matrix H'_P which has the above as its columns. Let $\Delta(p), p = 1, \ldots, P$ be the P rows of H'_P. Now set $\Delta_n(i) = \Delta(n \bmod P + 1), \forall n \geq 0, i \in S$. The perturbations are thus generated by cycling through the rows of H'_P. Here P is chosen as $P = 2^{\lceil \log_2(N+1) \rceil}$. Finally, matrices H_P for $P = 2^k$ are systematically constructed as follows:

$$H_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \text{ and } H_{2^k} = \begin{pmatrix} H_{2^{k-1}} & H_{2^{k-1}} \\ H_{2^{k-1}} & -H_{2^{k-1}} \end{pmatrix}, \ k > 1.$$

Suppose $\Delta_{i,j}$, $i \in \{1,\ldots,P\}$, $j \in \{1,\ldots,N\}$ corresponds to the (i,j)th element of the matrix H'_P. It is shown in [4] that for perturbations constructed as above,

$$\sum_{i=1}^{P} \frac{\Delta_{i,j}}{\Delta_{i,k}} = 0 \text{ and } \sum_{i=1}^{P} \frac{1}{\Delta_{i,l}} = 0, \ \forall j,k,l \in \{1,\ldots,N\}, \ j \neq k.$$

These properties are crucially required in establishing that the method asymptotically follows the steepest descent direction in the limit that the perturbation size (δ below) goes to zero, see [4] for details.

B. Algorithm-1

Let $Q_n(.,.)$ and $\hat{\pi}_i(n)$ denote the nth updates of the Q-function and randomized policy $\hat{\pi}_i, i \in S$, respectively. Let $\pi'_i(n) = \Gamma(\hat{\pi}_i(n) - \delta\Delta_n(i))$ where $\delta > 0$ is a given small constant. Suppose $\{a(n)\}$ and $\{b(n)\}$ are two step-size sequences that satisfy $a(n), b(n) > 0, \forall n \geq 0$ and

$$\sum_n a(n) = \sum_n b(n) = \infty, \ \sum_n (a(n)^2 + b(n)^2) < \infty, \ b(n) = o(a(n)), \tag{9}$$

respectively. Let

$$(\Delta_n(i))^{-1} \stackrel{\triangle}{=} \left(\frac{1}{\Delta_n(i,u_i^1)}, \ldots, \frac{1}{\Delta_n(i,u_i^N)} \right)^T, \ \forall i \in S.$$

Let $\gamma_0(n) = 1 - \pi'_i(n).\bar{1}$ where $\bar{1} = (1,\ldots,1)$ is an N-vector with all its entries equal to one. Now let $\psi_n(j)$ be the action chosen from the set $U(j)$ according to the distribution $(\gamma_0(n), \pi'_j(n))$. Thus $\gamma_0(n)$ corresponds to the probability of picking action $u_j^0 \in U(j)$, while $\pi'_j(n)$ is the vector of probabilities of picking the remaining actions in $U(j)$. Also, let $\eta_n(i,u)$ be as in (7).

Now for all $i \in S, u \in U(i)$, initialize $Q_0(i,u) = 0$ and $\hat{\pi}_i(0) = \left(\frac{1}{N+1}, \ldots, \frac{1}{N+1}\right)$, respectively. Then $\forall i \in S, u \in U(i)$,

$$Q_{n+1}(i,u) = Q_n(i,u) + b(n)\left(g(i,u,\eta_n(i,u)) + \alpha Q_n(\eta_n(i,u), \psi_n(\eta_n(i,u))) - Q_n(i,u)\right) \tag{10}$$

$$\hat{\pi}_i(n+1) = \Gamma\left(\hat{\pi}_i(n) + a(n)\frac{Q_n(i,\psi_n(i))}{\delta}(\Delta_n(i))^{-1}\right) \tag{11}$$

We thus update the randomized policy on the faster timescale and the Q-function on the slower one.

C. Algorithm-2

This algorithm is similar to Algorithm-1, except that we do not update Q-values for each state-action pair as in (10). Instead, Q-value updates are performed for states with actions picked according to the 'current' randomized policy update. Also, recursion (11) is exactly the same as before and is not written to save space. Thus, (10) is replaced by the following recursion:

$$Q_{n+1}(i,\hat{\psi}_n(i)) = Q_n(i,\hat{\psi}_n(i))+b(n)(g(i,\hat{\psi}_n(i),\eta_n(i,\hat{\psi}_n(i)))+\alpha Q_n(\eta_n(i,\hat{\psi}_n(i)),\psi_n(\eta_n(i,\hat{\psi}_n(i))))$$
$$-Q_n(i,\hat{\psi}_n(i))). \tag{12}$$

Here $\hat{\psi}_n(i)$ is the action chosen from the set $U(i)$ according to the distribution $(\beta_0(n), \hat{\pi}_i(n))$, where $\beta_0(n)$ is the probability of picking action $u_i^0 \in U(i)$ and is obtained from $\hat{\pi}_i(n)$ as $\beta_0(n) = 1 - \hat{\pi}_i(n).\bar{1}$. Also, $\psi_n(.), \eta_n(.,.)$ are as in Algorithm-1. Note that it is important for proper convergence behaviour that actions here are sampled as per the 'running' randomized policy estimate given by (11) (as $\hat{\psi}_n(i)$ does) and not according to any other distribution.

Note that Q-learning suffers from the problem of *lack of sufficient exploration* [14] that accentuates when the numbers of states and actions are large. An exploration step is usually recommended with Q-learning to alleviate this problem whereby actions that do not correspond to the 'minimizing action' are picked with some small probability. Since our algorithms work with stationary randomized probabilities, an additional exploration step is not required as actions other than the 'minimizing action' are picked with probabilities obtained from the randomized policy updates.

III. Sketch of Convergence

The analysis proceeds along standard lines and is therefore briefly sketched. We first consider Algorithm-1. Let $\mathcal{F}_n = \sigma(Q_j(i,u), \eta_j(i,u), \hat{\pi}_i(j), \psi_j(i),\ i \in S, u \in U(i), j \leq n)$ denote an increasing sequence of σ-fields. One can show in a similar manner as Theorem 2.1 of [15] that $\limsup_n \|Q_n(i,u)\| < \infty\ \forall i \in S, u \in U(i)$. Define sequences $\{M_n(i)\}, i \in S$, according to

$$M_n(i) = \sum_{j=0}^{n-1} a(j)\left(Q_j(i,\psi_j(i)) - E[Q_j(i,\psi_j(i))|\mathcal{F}_{j-1}]\right).$$

Then it can be seen using (9) that $\{M_n(i), \mathcal{F}_n\}$ are almost surely convergent martingale sequences. Hence the faster timescale recursion (11) can be written as

$$\hat{\pi}_i(n+1) = \Gamma\left(\hat{\pi}_i(n) + a(n)\frac{E[Q_n(i,\psi_n(i))|\mathcal{F}_{n-1}]}{\delta}(\Delta_n(i))^{-1} + \epsilon_1(n)\right) \tag{13}$$

where $\epsilon_1(n) = o(1)$ because of the above. Further, recursion (10) can be written as

$$Q_{n+1}(i,u) = Q_n(i,u) + a(n)\epsilon_2(n) \tag{14}$$

where, as a consequence of (9), $\epsilon_2(n) \to 0$ as $n \to \infty$. Thus, when viewed from the timescale corresponding to $\{a(n)\}$, recursion (10) can be seen to asymptotically track the trajectories of the ordinary differential equation (ODE):

$$\dot{Q}_t(i,u) = 0, \ \ i \in S, u \in U(i). \tag{15}$$

Note that by (15), $Q_t(i,u)$ are time-invariant when viewed from the faster timescale; hence we suppress the index t and denote by $Q(i,u)$ the above. Now, in (13), suppose that $Q_j^{\pi_i'(j)}(i) \stackrel{\triangle}{=} E[Q_j(i,\psi_j(i))|\mathcal{F}_{j-1}]$. Using a Taylor series expansion of $Q_j^{\pi_i'(j)}(i)$ around $\hat{\pi}_i(j)$ and the claim in Corollary 2.6 of [4], it can be seen that recursion (11) asymptotically tracks, in the limit as $\delta \to 0$, the trajectories of the ODE

$$\dot{\hat{\pi}}_i = \hat{\Gamma}(-\nabla Q^{\hat{\pi}_i(t)}(i)). \tag{16}$$

In the above, $\hat{\Gamma}(.)$ is an operator that restricts the ODE (16) to evolve within the simplex PS and is defined according to

$$\hat{\Gamma}(v(y)) = \lim_{\gamma \downarrow 0} \left(\frac{\Gamma(y + \gamma v(y)) - \Gamma(y)}{\gamma} \right),$$

for any bounded, continuous $v(.)$. The stable fixed points of (16) lie within the set $M = \{\hat{\pi}_i | \hat{\Gamma}(\nabla Q^{\hat{\pi}_i}(i)) = 0\}$. It can be seen that $V_{\hat{\pi}} = \sum_{i \in S} Q^{\hat{\pi}_i}(i)$ serves as a strict Liapunov function for the ODE (16). Let $Q^*(i,u)$ correspond to the unique solution of (4) $\forall i \in S, u \in U(i)$. Using again a similar martingale type argument as before on the slower timescale recursion (11), we obtain

Theorem 1: For all $i \in S, u \in U(i)$, the quantities $Q_n(i,u)$ as given by Algorithm-1 converge to $Q^*(i,u)$ in the limit as $\delta \to 0$.

Finally, note that a similar analysis can be shown for Algorithm-2 as well, except that since $\hat{\psi}_n(i)$ simulate the actions to be chosen in state i (cf.(12)), $Q_n(i, \hat{\psi}_n(i))$ will converge to the unique solution of (8) in place of (4).

IV. Numerical Experiments

We consider networks with different topologies involving 4, 8 and 16 nodes, respectively. We show here results of experiments with networks having 4 and 16 nodes, respectively, as similar results were obtained for the 8–node case. We compare the performance of our algorithms on these settings with the regular Q-learning algorithm. In all cases studied, 0 is the source and the node with highest number is the destination. The network configurations are shown in Figs. 1 and 2, respectively. We assume that all links are bidirectional and have the same cost values along both directions. For ease of exposition, we assign numbers $0, 1, 2$ etc., to the links emerging from the various nodes, see Figs. 1 and 2. Thus, for instance in Fig. 2, link 0 at node 4 corresponds to the same link as link 2 at node 1 (but in the reverse direction). The action set $U(i)$ at node i corresponds to the set of links connected to that node. Selecting an action at a node thus corresponds to selecting a link at that node for routing packets. We denote by $p(x, j)$, the probability of selecting link j at node x. Further, we denote by $(n_1 - n_2 - \cdots - n_{l-1} - n_l)$ a path from node n_1 to node n_l through intermediate nodes $n_2, \ldots, n_{l-1}$.

A. Network with 4 Nodes

We consider here a network with four nodes and six bidirectional links. The network configuration is shown in Fig. 1. In these experiments, we choose $\alpha = 0.9$, the step-size sequences are chosen as $a(n) = 1/n$, $b(n) = \frac{1}{n^\beta}, \forall n \geq 1$, with $a(0) = b(0) = 1$, and β is set at 0.7. Further, δ was set at 0.06. When the one-stage costs are high in magnitude, normalizing these is seen to improve performance. The network learns quickly to route packets along the shortest path. The link costs are varied manually to obtain different predefined optimal paths between source and destination, and tested using the algorithms. The results obtained using Algorithms 1 and 2 for the case when the optimal path corresponds to $(0 - 1 - 2 - 3)$ are shown in Tables I and II, respectively. Here $p(x, i)$ and $Q(x, i)$, respectively, denote the transition probabilities and Q-values obtained for the node-link pair (x, i). Also, Table III shows the corresponding Q-values obtained using the Q-learning algorithm. The values shown in the above tables are obtained after 50,000 updates of the corresponding algorithms. We observe however that convergence of all three algorithms takes place in much less number of updates (less than 5000 updates in most cases). In Figs. 3 and 4, we show plots of probability and Q-value updates with the number of iterations using Algorithms 1 and 2, respectively, for node 0. We show these plots only for

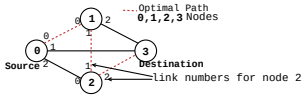

Fig. 1. Network with 4 nodes

2000 updates to give some idea of the relative convergence trends for the two algorithms. We found that in all cases and network topologies that we studied, convergence is achieved faster when Algorithm-2 is used (over Algorithm-1). We also varied link costs manually to obtain the optimal/shortest path configurations $(0-3), (0-2-1-3), (0-1-3)$ and $(0-2-3)$, respectively. Our algorithms converged to the optimal path configurations in all cases in the manner as above.

B. Network with 16 *Nodes*

The network configuration is shown in Fig. 2. We consider here a network with 16 nodes and 24 bidirectional links. The results obtained for the optimal path setting $(0-1-4-8-12-14-15)$ using both algorithms are shown in Tables IV and V, respectively. The Q-values obtained using the Q-learning algorithm are shown in Table VI. We also performed simulations using the algorithms after varying link costs manually to obtain the optimal path configurations $(0-2-4-8-11-14-15)$, $(0-1-3-6-10-13-15)$ and $(0-2-5-9-12-14-15)$, respectively. As before, we ran simulations for $50,000$ iterations in all cases during which convergence to the above optimal path configurations using our algorithms had been achieved. However, as with the previous network configuration involving 4 nodes, the algorithms converged here as well in far less number of iterations. On a Pentium IV computer with 2.4 GHz processor, it took less than one minute to complete the simulation for 50,000 iterations in all cases. The codes for the various network topologies were written using the C programming language. On a network with 16 nodes and for $50,000$ iterations, the amount of CPU time taken by Algorithm-1 was found to be 1.835 seconds, by Algorithm-2 was 1.141 seconds and by the usual Q-learning algorithm was 1.029 seconds. In general (over the various settings tried), Algorithm-2 was found to converge faster than Algorithm-1 and was comparable (in speed) to the Q-learning algorithm.

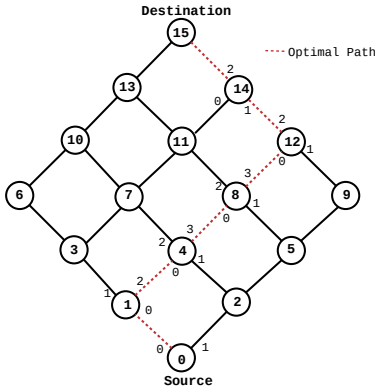

Fig. 2. Network with 16 nodes

TABLE I

CONVERGED PROBABILITY VECTORS AND Q-VALUES FOR OPTIMAL PATH $(0-1-2-3)$ USING ALGORITHM-1

Node(x)	p(x,0)	p(x,1)	p(x,2)	Q(x,0)	Q(x,1)	Q(x,2)
0	0.8082	0.1133	0.0783	0.5498	1.1174	1.0000
1	0.2065	0.5303	0.2631	0.6889	0.2242	1.0000
2	0.0242	0.0039	0.9717	1.5865	0.5414	0.1000

TABLE II

CONVERGED PROBABILITY VECTORS AND Q-VALUES FOR OPTIMAL PATH $(0-1-2-3)$ USING ALGORITHM-2

Node(x)	p(x,0)	p(x,1)	p(x,2)	Q(x,0)	Q(x,1)	Q(x,2)
0	0.9191	0.0052	0.0756	0.3508	1.1478	0.9779
1	0.1508	0.8398	0.0092	0.4771	0.2249	0.8876
2	0.0099	0.1330	0.8570	1.2577	0.3442	0.1000

TABLE III

CONVERGED Q-VALUES FOR OPTIMAL PATH $(0-1-2-3)$ USING REGULAR Q-LEARNING ALGORITHM

Node(x)	Q(x,0)	Q(x,1)	Q(x,2)
0	0.2710	1.0900	1.0000
1	0.3439	0.1900	1.0000
2	1.2439	0.2710	0.1000

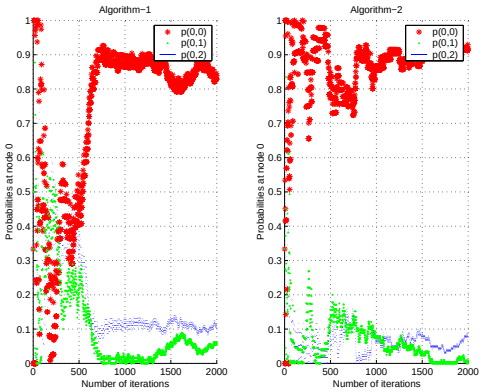

Fig. 3. Plot of Probability Updates vs. Number of Iterations at Node 0 for Optimal Path $(0-1-2-3)$

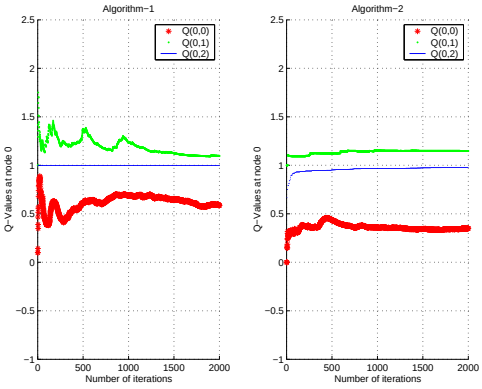

Fig. 4. Plot of Q-factors vs. Number of Iterations at Node 0 for Optimal Path$(0-1-2-3)$

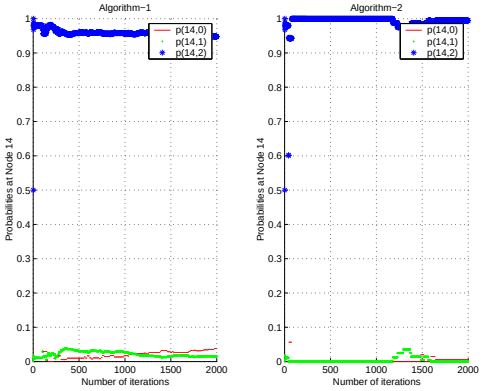

Fig. 5. Plot of Probability Updates vs. Number of Iterations at Node 14 for Optimal Path $(0-1-4-8-12-14-15)$

TABLE IV

CONVERGED PROBABILITY VECTORS AND Q-VALUES FOR OPTIMAL PATH (0-1-4-8-12-14-15) USING ALGORITHM-1

Node(x)	**p(x,0)**	**p(x,1)**	**p(x,2)**	**p(x,3)**	**Q(x,0)**	**Q(x,1)**	**Q(x,2)**	**Q(x,3)**
0	0.9570	0.0429	-	-	0.1215	0.1612	-	-
1	0.0543	0.3193	0.6263	-	0.1135	0.1653	0.1161	-
4	0.1094	0.4015	0.0627	0.4262	0.1211	0.1610	0.1487	0.0836
8	0.0073	0.1298	0.3755	0.4872	0.1146	0.1393	0.1032	0.0642
12	0.1302	0.1701	0.6995	-	0.0858	0.1235	0.0510	-
14	0.0333	0.0132	0.9535	-	0.1045	0.0656	0.0033	-

TABLE V

CONVERGED PROBABILITY VECTORS AND Q-VALUES FOR OPTIMAL PATH (0-1-4-8-12-14-15) USING ALGORITHM-2

Node(x)	**p(x,0)**	**p(x,1)**	**p(x,2)**	**p(x,3)**	**Q(x,0)**	**Q(x,1)**	**Q(x,2)**	**Q(x,3)**
0	0.8972	0.1027	-	-	0.2442	0.3516	-	-
1	0.0316	0.2603	0.7079	-	0.3966	0.3360	0.2331	-
4	0.2236	0.0188	0.2827	0.4747	0.2494	0.6295	0.3055	0.1705
8	0.0700	0.0932	0.1094	0.7271	0.3029	0.2977	0.2951	0.1460
12	0.2026	0.0258	0.7715	-	0.1940	0.3837	0.1261	-
14	0.0175	0.0000	0.9825	-	0.2592	0.1545	0.0050	-

TABLE VI

CONVERGED Q-VALUES FOR OPTIMAL PATH $(0-1-4-8-12-14-15)$ USING REGULAR Q-LEARNING ALGORITHM

Node(x)	**Q(x,0)**	**Q(x,1)**	**Q(x,2)**	**Q(x,3)**
0	0.4685	2.1785	-	-
1	0.5217	2.2317	0.4095	-
4	0.4685	2.1785	2.1785	0.3439
8	0.4095	2.1195	1.9810	0.2710
12	0.3439	2.0539	0.1900	-
14	1.9810	0.2710	0.1000	-

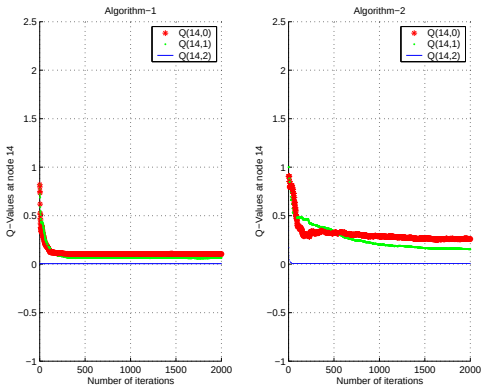

Fig. 6. Plot of Q-factors vs. Number of Iterations at Node 14 for Optimal Path $(0-1-4-8-12-14-15)$

V. Conclusions

In this note, we developed two gradient search based variants of the Q-learning algorithm that both use two-timescale stochastic approximation and studied applications of these to the problem of routing in communication networks. One of the algorithms (Algorithm-1) updates Q-values associated with each state-action pair while the other one (Algorithm-2) updates Q-values of states with actions chosen according to 'current' randomized policy updates. Both our algorithms pick 'minimizing actions' for the Q-value updates using probabilities obtained from the randomized policy updates. Thus actions other than the 'minimizing action(s)' are also picked with certain probabilities that however asymptotically diminish. Our algorithms therefore do not suffer from problems due to lack of sufficient exploration as with regular Q-learning. We gave a sketch of convergence of the algorithms. We finally showed numerical experiments for a problem of routing on different network topologies and with different optimal path configurations. Both our algorithms were found to converge to the optimal path configurations. We did not use an extra 'exploration step' in the implementation of the regular Q-learning algorithm here. This may however become necessary in cases when the numbers of states and actions are large. Algorithm-2 may have computational advantages in such scenarios. We considered synchronous implementations for all three algorithms. Our algorithms could suitably be modified and implemented on settings involving distributed implementations with asynchronous updates on multiple processors with communication/feedback delays [15]. Finally, we did not consider settings with dynamically varying link costs and/or topologies as with real network scenarios. The algorithms

need to be tested on such settings as well. Further computational experiments are needed to determine if our algorithms are useful for large scale networks.

REFERENCES

[1] Barto, A., Sutton, R. and Anderson, C. (1983) "Neuron-like elements that can solve difficult learning control problems", *IEEE Transactions on Automatic Control*, 13:835–846.

[2] Bertsekas, D.P. and Tsitsiklis, J.N. (1996) "Neuro-Dynamic Programming", Athena Scientific, Belmont, MA.

[3] Bertsekas, D.P. (1995) "Dynamic Programming and Optimal Control", Athena Scientific, Belmont, MA.

[4] Bhatnagar, S., Fu, M.C., Marcus, S.I. and Wang, I.-J. (2003) "Two-timescale simultaneous perturbation stochastic approximation using deterministic perturbation sequences," *ACM Transactions on Modelling and Computer Simulation*, 13(4):180–209.

[5] Bhatnagar, S. and Kumar, S. (2004) "A simultaneous perturbation stochastic approximation-based actor-critic algorithm for Markov decision processes," *IEEE Transactions on Automatic Control*, 49(4):592–598.

[6] Borkar, V.S. (1997) "Stochastic approximation with two time scales", *Systems Control Lett.*, 29:291-294.

[7] Boyan, J.A. and Littman, M.L. (1994) "Packet routing in dynamically changing networks: a reinforcement learning approach", *Advances in Neural Information Processing Systems*, 6:671–678.

[8] Konda, V.R. and Borkar, V.S. (1999) "Actor-critic like learning algorithms for Markov decision processes", *SIAM J. Control Optim.*, 38(1):94–123.

[9] Kumar, S. and Miikkulainen, R. (1997) "Dual reinforcement Q-routing: an on-line adaptive routing algorithm", *Proceedings of Intelligent Engineering Systems Through Artificial Neural Networks (ANNIE-97), St. Louis, MO*, 7:231–238.

[10] Marbach, P., Mihatsch, O. and Tsitsiklis, J.N. (2000) "Call admission control and routing in integrated services networks using neuro-dynamic programming", *IEEE Journal on Selected Areas in Communication*, 18:197–208.

[11] Spall, J.C. (1992) "Multivariate stochastic approximation using a simultaneous perturbation gradient approximation," *IEEE Transactions on Automatic Control*, 37:332–341.

[12] Spall, J.C. (1997) "A one-measurement form of simultaneous perturbation stochastic approximation", *Automatica*, 33:109–112.

[13] Subramanian, D., Druschel, P. and Chen, J. (1997) "Ants and reinforcement learning: a case study in routing in dynamic networks", *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI-97)*, 832–839.

[14] Sutton, R. and Barto, A. (1998) "Reinforcement Learning: An Introduction", MIT Press, Cambridge, MA.

[15] Tsitsiklis, J.N. (1994) "Asynchronous stochastic approximation and Q-learning", *Machine Learning*, 16:185–202.

[16] Watkins, J.C.H. and Dayan, P. (1992) "Q-learning", *Machine Learning*, 8:279–292.